System Definition Guide

Vitech

Vitech Corporation

2070 Chain Bridge Road, Suite 100
Vienna, Virginia 22182-2536
703.883.2270 FAX: 703.883.1860
Customer Support: support@vitechcorp.com
www.vitechcorp.com

Publication Date: September 2007

Table of Contents

List of Figures

List of Tables

Preface

This guide provides a structured approach to populating a CORE project with system engineering information using the basic schema provided with CORE. It presents actions that must be accomplished in the context of the classic system engineering activities of requirements analysis, functional analysis, physical architecture synthesis, and verification and validation as illustrated in Figure 1. Thus, the approach is consistent with commonly used system engineering handbooks and standards, company unique Capability Maturity Model® Integration (CMMI) processes, and project specific System Engineering Plans (SEP) and System Engineering Management Plans (SEMP).

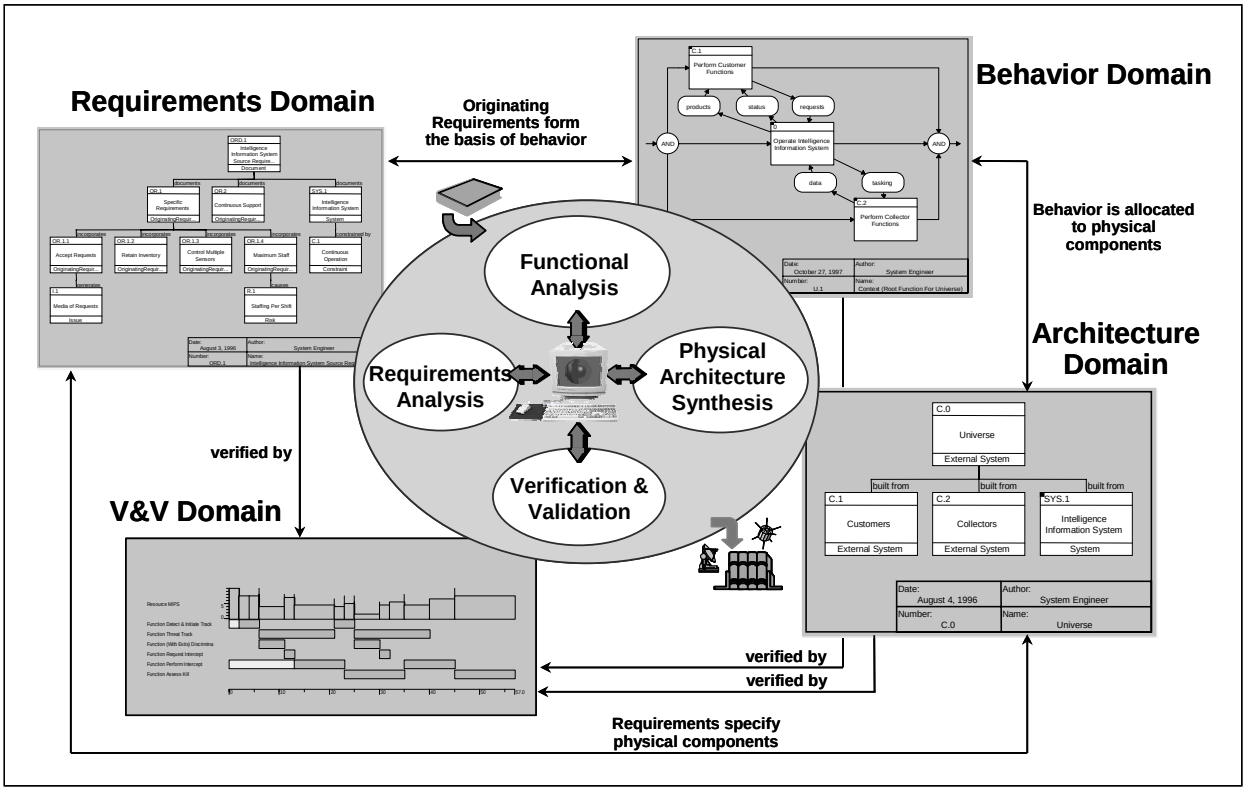

Figure 1 System Engineering Activities

Each activity and the CORE schema classes used to capture the associated information in the database are described. The activity discussion is followed by a schema diagram and table identifying the schema classes to be used and the attributes and relationships that should be established when performing the activity. In addressing each activity, attention is given to populating the database in a manner that facilitates the production of formal documents using the standard scripts provided with CORE. This guide describes the development of a system design that results in standard documentation at minimal additional cost.

This guide is intended to augment the Model-Based System Engineering with CORE® training course with reference material to help the student retain what was learned in the training class and make the most effective use of CORE on a project. The approach is generic and is not exhaustive of all cases and situations. The approach is written in the context of top-down system engineering. The activities can be reordered for reverse engineering.

The following additional resources are available for use with this guide:

- For descriptions of different views, behavior diagram notation, and the mechanics of entering data into CORE, the reader is referred to the CORE on-line help.
- For the definition of schema terms, the reader is referred to the CORE schema which contains descriptions for each schema entity and to the Schema Definition Report script which outputs documentation of the schema definition for a selected facility.
- For details on generating standard formal documents, the reader is referred to the script help file for the formal documentation scripts. This documentation can be opened by selecting the Run Script icon on the CORE Explorer toolbar, selecting any one of the formal documentation scripts such as the System/Segment Specification (SSS), and pressing the Script Help File button.

1 Requirements Capture

This section is written assuming that the customer or end-user has provided a system requirements specification. If that is not the case, it is assumed that system engineering will start with the task of collecting all stakeholder needs and transforming them into required functionality and performance and design constraints. The end result of this effort will be a collection of requirements that are treated as originating requirements (See Section 1.2).

1.1 Define Need and System Concept

Identify the system and its mission. Physical entities, including the system, external systems, and entities within the system (excluding interfaces) are represented in CORE as elements in the class **Component**. When creating the **Component** that represents the system, the attributes listed in the Table 1 should be assigned values. A **Component's** Type attribute designates what the element represents (in this case a System).

Table 1 System Definition

Element Class	Attributes	Relationships	Target Classes
Component	Abbreviation Description Doc. PUID Mission Number Purpose Type: System		

1.2 Capture Source (Originating) Requirements

Capturing requirements from source documents involves the creation in the database of the following:

- **Document** element for each source document
- **Requirement** element for each source requirement
- **ExternalFile** element for each source requirement-related table or graphic
- **DefinedTerm** element for each pertinent acronym or special term in the source requirements

As part of the extraction, the following should be done:

- Parse compound requirements into single, testable **Requirements** statements. These should be linked to their parent **Requirement** using the *refines/refined by* relationship.
- Place any requirements tables and graphics in separate files and reference them in the project database using **ExternalFile** elements where each *augments* the subject **Requirement**. The formal documentation scripts, as well as the System Description Document (SDD) script, will automatically include these external tables and graphics

in the output immediately following the element Description and make entries in the List of Figures and List of Tables, as appropriate. In order to properly number and label the tables and graphics for inclusion in the output, only a single graphic or table should appear in each file.

- Acronyms and/or special terms appearing in the source document should be captured in the database as **DefinedTerms**. For an acronym or abbreviation, the acronym is entered into the Acronym attribute and what it stands for is entered as the name of the element. For a special term, the term is the name of the element and its definition is entered into the Description attribute. By filling in both the Acronym and Description attributes, appropriate entries will appear in both the acronym and glossary sections of documents generated using the CORE formal documentation scripts once the **DefinedTerm** is linked to the output **Document** using the *used by* relationship.

The following paragraphs contain information on special topics concerning the entry of source requirements.

Extracting requirements from source documents. The entry of source requirements into a CORE database may be accomplished by using one or more of the following:

- Element Extractor window
- Document/Shall Parser script
- Advanced CSV File Parser script if the requirements are being transferred as a CSV file from another application such as Telelogic DOORS, Microsoft Excel, or Microsoft Access
- Copy and Paste or Paste Unformatted commands.

Setting the Origin attribute. It is important to determine the customer's acceptable requirements traceability. A customer may require traceability to the exact wording in a source document or may allow traceability to parsed statements. Once this decision has been made, set the Origin attribute to Originating for each **Requirement** in the document requirements hierarchy down through the lowest-level traceable **Requirements** (i.e., those deemed originating by the customer). For all other **Requirements**, set the Origin attribute to Derived. This will record which **Requirements** are originating and ensure that the traceability matrix produced by the CORE formal documentation scripts traces to the correct source **Requirement** elements.

Establishing Doc. PUIDs. If the source documents have assigned Project-Unique Identifiers (PUIDs) to the requirements, these should be captured in the database in the Doc. PUID attribute of the **Requirement**. If PUIDs have not been pre-assigned, it is advisable to assign one to each originating **Requirement**. This can be done manually or by running the Assign Documentation PUID utility script.

Note: *If available, Doc. PUIDs are automatically output by all of the formal documentation scripts. To take advantage of this feature, Doc. PUIDs should be assigned to elements in the following classes:* **Component**, **Function**, **Interface**, **Item**, **Link**, **Requirement**, *and* **VerificationRequirement**.

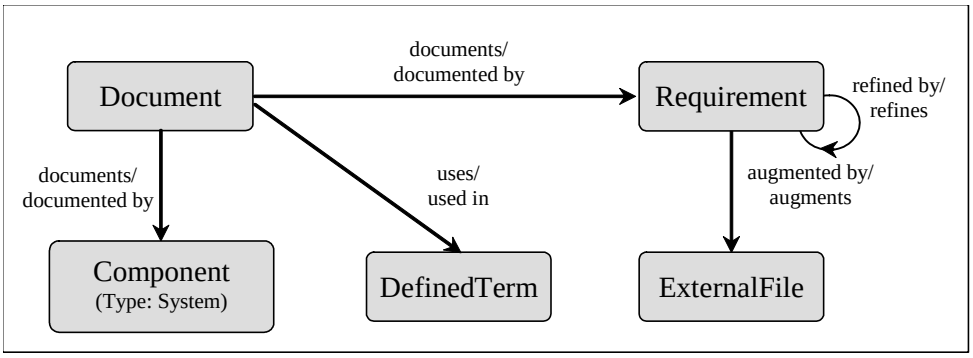

Figure 2 Source Requirements

Table 2 Source Requirements

Element Class	Attributes	Relationships	Target Classes
DefinedTerm	Acronym Description	*used in / uses*	**Document**
Document	Description Document Date Document Number External File Path either[1] Govt. Category or Non-Govt. Category Number Type	*documents / documented by*[2]	**Component** (Type: System) **Requirement**
		uses / used in	**DefinedTerm**
ExternalFile	Description External File Path Number Page Orientation Title Type	*augments / augmented by*[3]	**Requirement**
Requirement	Description	*augmented by / augments*[3]	**ExternalFile**

[1] These attributes are used when the source document is to be listed as an applicable document in a formal document generated from the database. See Section 1.4 for an explanation.

[2] Only the top-level **Requirements** need to be *documented by* the source **Document**. The formal documentation scripts search up the requirements hierarchy to locate the source **Document.**

[3] The Position attribute of this relationship should be set to control the order in which multiple external files are appended to the **Requirement's** Description attribute when it is output in formal documentation generated from the database.

Table 2 Source Requirements

Element Class	Attributes	Relationships	Target Classes
	Doc. PUID Key Performance Parameter Number Origin: Originating Paragraph Number[4] Paragraph Title[4] Rationale Units Value Weight Factor	*documented by / documents*	**Document**
		refined by / refines	**Requirement**
		refines / refined by	**Requirement**

Warning: *The default font for text attributes, such as Description, is Times New Roman 10. Within a text attribute, the user has control over color, fonts, styling, sizing, and special effects such as underline, superscript, and strikethrough. The formal documentation scripts do not override any user modified fonts or special effects; however, they can override color, styling, and font size if the font is Times New Roman (they only control the styling of text in Times New Roman). Consequently, in order to produce professional looking documents, care should be taken when capturing external source material. Specifically, when using the Element Extractor window, either turn off the Maintain Formatting option or pre-process the document to convert all text to Times New Roman (i.e., open the document in a word processor, select all contents of the document, and select Times New Roman as the font). Similarly, when using cut & paste, either pre-process the document to set the font to Times New Roman or use Paste Unformatted rather than the Paste command. Since they should not be modified on output, formulas should be captured in another font, such as Arial. Also, note that text attributes do not support embedded tables and graphics. Therefore, tables and graphics should be captured as* ***ExternalFile*** *elements.*

1.3 Define System Boundary

Based on an examination of the originating requirements, identify the system boundary and context. To define the boundary, identify each external with which the system must interface. An external is represented as a **Component** and may identify the system environment, an actual external system, or a human. Create a **Component** element representing the context and decompose it into the system and its externals using the *built from* relationship. Set the Type attribute for each **Component**. Note that humans may be considered as part of the system or as external to the system depending on the actions they take or the role that they play in performing the system actions. In many cases, there are humans that are part of the system and humans that are external to the system.

[4] Used to record the source document paragraph number and title for an originating **Requirement**.

To complete the system boundary definition, identify all interfaces between the system and each external by creating elements of the **Interface** class. Defining an **Interface** element establishes that the system interacts with an external. Typically, there will be only one interface between the system and each external. The details of the interface are documented by **Link** elements (See Section 4.2).

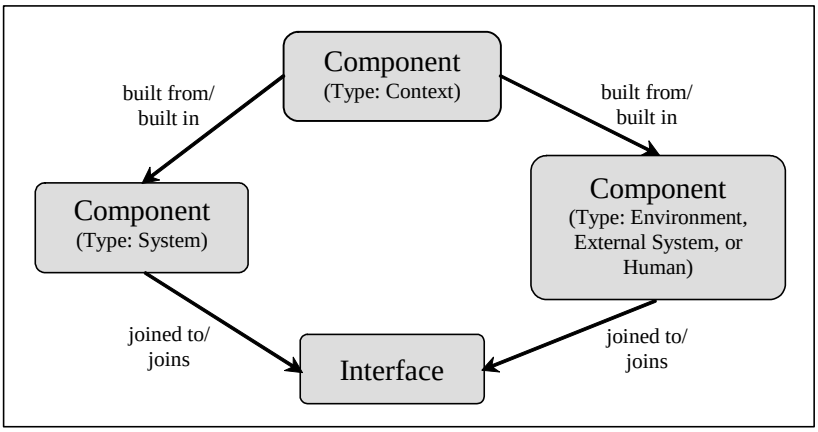

Figure 3 System Boundary

Table 3 System Boundary

Element Class	Attributes	Relationships	Target Classes
Component (Type: Context)	Description Number Type: Context	*built from / built in*	**Component** (Type: System and Environment, External System, or Human)
Component (Type: Environment, External System, or Human)	Abbreviation Description Number Type: Environment, External System, or Human	*built in / built from*	**Component** (Type: Context)
		joined to / joins	**Interface**
Component (Type: System)	See Section 1.1	*built in / built from*	**Component** (Type: Context)
		joined to / joins	**Interface**
Interface	Description Doc. PUID Number	*joins / joined to*	**Component** (Type: System and Environment, External System, or Human)

Suggestion: *Create a folder for the context and externals in order to separate them from the evolving system component hierarchy. Typically, the context and externals are given a different numbering scheme than the elements in the system component hierarchy in order to differentiate them in CORE views such as the Physical Block Diagram and Hierarchy diagrams.*

1.4 Collect Additional Applicable Documents

Identify any other applicable or reference documents such as standards, regulatory documents, and Interface Control Documents for interfaces to existing external systems. These or specific portions of these may be referenced in the source requirements. If a reference is not sufficient, extract additional **Requirements** from the applicable or reference documents.

Note: *The applicable documents will eventually be linked to other* **Document** *elements that represent the formal documents generated from the database. The formal documentation scripts will automatically include the* references *to applicable documents in Section 2 of the produced specification or design document once these relationships are established. In order for an applicable document to appear in the correct subsection, either the Govt. Category or the Non-Govt. Category attribute must be set to the appropriate subsection heading.*

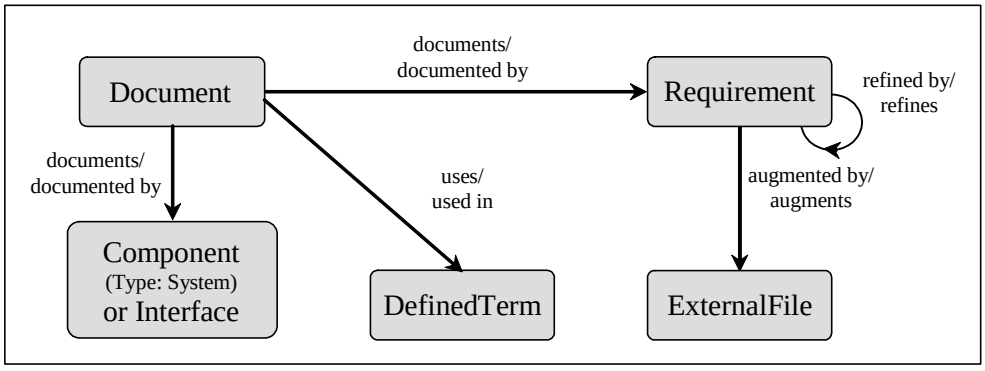

Figure 4 Applicable Documents and Requirements

Table 4 Applicable Documents and Requirements

Element Class	Attributes	Relationships	Target Classes
DefinedTerm	See Section 1.2	*used in / uses*	**Document**
Document	See Section 1.2	*documents / documented by*	**Component** **Interface** **Requirement**
		uses / used in	**DefinedTerm**
ExternalFile	See Section 1.2	*augments / augmented by*[5]	**Requirement**
Interface	See Section 1.3	*documented by / documents*	**Document**
Requirement	See Section 1.2	*augmented by / augments*[5]	**ExternalFile**
		documented by / documents	**Document**
		refined by / refines	**Requirement**
		refines / refined by	**Requirement**

Suggestion: *Create folders to group source documents and applicable documents. An element, such as a* **Document**, *may appear in multiple folders. To remove an element from a folder, drag it to another folder. Deleting an element removes the element from the database.*

[5] The Position attribute of this relationship should be set to control the order in which multiple external files are appended to the requirement description when it is output in formal documentation generated from the database.

2 Requirements Analysis

Requirements analysis involves a collection of concurrent, inter-related activities. These are addressed in the following subsections.

2.1 Parse Originating Requirements

If not previously done when capturing source requirements (See Section 1.2), parse the originating **Requirements** into single, testable **Requirements** statements. This parsing can result in issues to be resolved. These should be identified as described in Section 2.2 below. See Section 1.2 for a discussion of Originating vs. Derived **Requirements**.

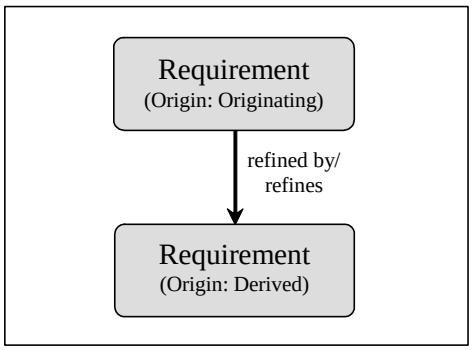

Figure 5 Derived Requirements

Table 5 Derived Requirements

Element Class	Attributes	Relationships	Target Classes
Requirement	Description Doc. PUID Number Origin: Derived Rationale Weight Factor	*refined by / refines*	**Requirement**
		refines / refined by	**Requirement**

2.2 Identify Requirement Issues and Risks

Requirement Issues. Examine each parsed source **Requirement**, capturing any questions or problems identified by creating **Issue** elements. The assignment of resolution responsibility to an individual or organization is captured by the *assigned to* relationship between the **Issue** and an **Organization** element. The resolution of an **Issue** may require customer involvement and/or trade studies. These should be captured in the database using the **Document** element and linked to the **Issue** using the *documented by* relation. The resolution of an **Issue** is not a requirement but generally either *results in* a derived or design decision **Requirement,** or the addition or clarification of **Requirements**. Any resultant **Requirement** should be linked to both the **Issue** and the **Requirement(s)** that generated the **Issue**.

Requirement Risks. Risks are issues that are significant enough to potentially affect the achievement of a major program objective or milestone. Because the information needed is

different than that of an Issue, **Risk** is a separate element class in CORE. Among the many sources of risk on a program, one of these is the requirements themselves. Therefore, examine each leaf-level source **Requirement** and identify any **Risks**. This may be done by the system engineers or risk management personnel depending on the project organization. Generally, **Risks** are addressed by a **ProgramActivity** or **ProgramElement** in the Program Management Facility. Details of program management are not addressed in this guide. In addition, **Risks** may *result in* new **Requirements**. Any risk status graphs should be identified as **ExternalFiles** and linked to the **Risk** using the *augments* relationship. Similarly, any risk status reports should be identified as **Documents** and linked to the **Risk** using the *documents* relationship.

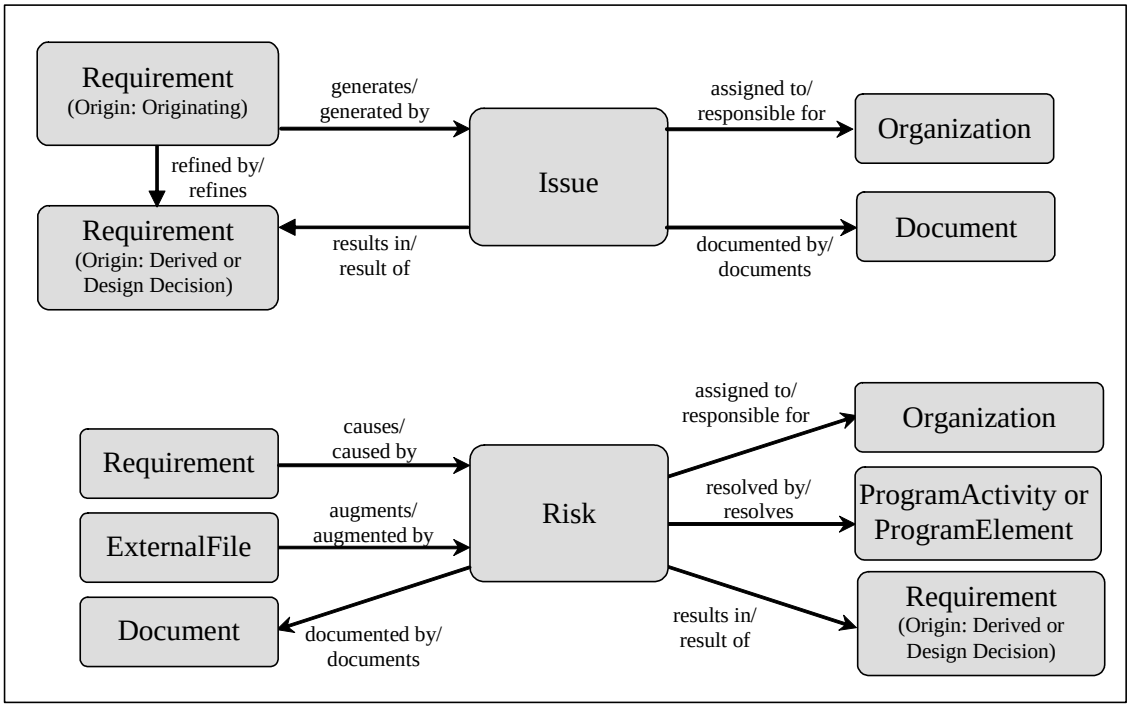

Figure 6 Requirement Issue and Risk

Table 6 Requirement Issue and Risk

Element Class	Attributes	Relationships	Target Classes
Document	Description Document Date Document Number External File Path Number Type	*documents / documented by*	**Issue** **Risk**
ExternalFile	See Section 1.2	*documents / documented by*	**Issue** **Risk**
Issue	Alternatives Assumptions Decision Description Due Date Number Originator Rationale Severity Status	*assigned to / responsible for*	**Organization**
		documented by / documents	**Document**
		generated by / generates	**Requirement**
		results in / result of	**Requirement**
Organization	Abbreviation Description Number Role	*responsible for / assigned to*	**Issue** **Risk**
ProgramActivity	Description End Date Number Start Date	*resolves / resolved by*	**Risk**
ProgramElement	Description End Date Number Start Date	*resolves / resolved by*	**Risk**
Requirement	See Sections 1.2 and 2.1	*causes / caused by*	**Risk**
		generates / generated by	**Issue**
		refined by / refines	**Requirement**
		refines / refined by	**Requirement**

Table 6 Requirement Issue and Risk

Element Class	Attributes	Relationships	Target Classes
		result of / results in	**Issue** **Risk**
Risk	Consequences Description Impact Likelihood Mitigation Plan Risk Factor Type	*assigned to / responsible for*	**Organization**
		augmented by / augments	**ExternalFile**
		caused by / causes	**Requirement**
		documented by/ documents	**Document**
		resolved by / resolves	**ProgramActivity** **ProgramElement**
		results in / result of	**Requirement**

2.3 Characterize Requirements and Categorize Constraints

Requirements can be characterized as one of the following:

- Functional (i.e., what the system must do)
- Performance (i.e., how well the system or function must perform)
- Constraint (i.e., limitation on the design or construction of the system)
- Verification (i.e., acceptance or test constraints).

This aspect of a requirement is captured in the database by setting the **Requirement's** Type attribute to the appropriate value. If a determination cannot be made, parse the **Requirement** into a set of **Requirements** where each **Requirement** is only one of the four types.

Link the system-level constraint **Requirements** to the system **Component** using the *specifies* relationship. As the system component hierarchy evolves, a constraint **Requirement** should be linked to all of the **Components** to which it applies (i.e., a constraint **Requirement** may apply to the descendants of a **Component** as well as the **Component**). See Section 4.4 for a discussion of constraints on lower-level **Components** and constraint hierarchies.

Constraint Categorization. For each **Requirement** that is a constraint, categorize it by a **Category** that represents the appropriate requirements domain, such as Reliability, Transportability, Electromagnetic Radiation, etc. These domains correspond to the non-functional leaf-level requirements sections typically found in a System/Segment Specification or other specification. For the standard formal documents supported by CORE, these **Categories** have been pre-defined in the Document Template RDT file provided with CORE. The templates are located in the Reports\Formal Documentation\Document Templates folder and should be imported into the database to obtain the suggested specification outlines (i.e., **Documents** and **Sections**), and associated **ReportScripts** and **Category** elements. See the formal documentation scripts help file for further information.

Note: *As further explained in the formal documentation script help file,* **Categories** *and the specification outlines should remain synchronized.* **Categories** *may be hierarchical; in which case, the hierarchy will control the definition of lower-level* **Sections** *in the formal documents. When using category hierarchies, only the leaf-level* **Categories** *should* categorize *requirements.*

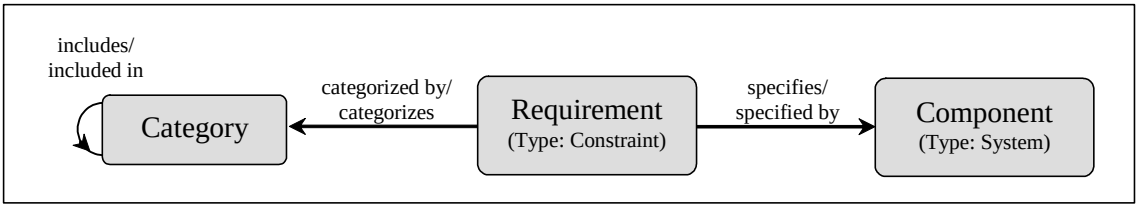

Figure 7 Constraint Requirements

Table 7 Constraint Requirements

Element Class	Attributes	Relationships	Target Classes
Category	Description	*categorizes / categorized by*	**Requirement** (Type: Constraint)
		included in / includes	**Category**
		includes / included in	**Category**
Component (Type: System)	See Section 1.1	*specified by / specifies*	**Requirement** (Type: Constraint)
Requirement	See Section 1.2 Type: Constraint	*categorized by / categorizes*	**Category**
		specifies / specified by	**Component** (Type: System)

3 Functional Analysis

3.1 Identify States/Modes (If Needed)

On some projects, either the customer expects or it is desirable from a documentation point of view to organize **Functions** by states and/or modes. Identify any states and/or modes of the system using **State/Mode** elements. Since states and/or modes could be used in the **Component** hierarchy at other than the system level and **Functions** will be mapped to **States/Modes** (See Section 3.2), separate **State/Mode** hierarchies need to be defined for each **Component** needing a state/mode representation. The Title attribute allows the formal documentation reference names to be the same for different **State/Mode** elements. A **State/Mode** can be *specified by* a **Requirement**. A state/mode transition diagram can be referenced in the database by creating an **ExternalFile** element that *augments* a **State/Mode** element.

Note: *Some organizations identify phases of operations of the system rather than states/modes. In this case, it is recommended that the* **State/Mode** *class be used to identify the phases and that the* **State/Mode** *class name be aliased to Phase.*

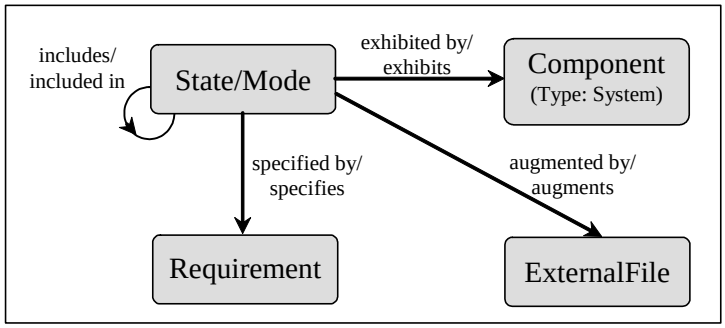

Figure 8 States and Modes

Table 8 States and Modes

Element Class	Attributes	Relationships	Target Classes
Component (Type: System)	See Section 1.1	*exhibits / exhibited by*[6]	**State/Mode**
ExternalFile	See Section 1.2	*augments / augmented by*[7]	**State/Mode**
Requirement	See Sections 1.2, 2.1, and 2.3	*specifies / specified by*	**State/Mode**
State/Mode	Description Number Title	*augmented by / augments*[7]	**ExternalFile**
		exhibited by / exhibits[6]	**Component**

[6] A component should link only to the top elements of a state/mode hierarchy. The formal documentation scripts will automatically include the entire state/mode hierarchy for the associated component.

[7] The Position attribute of this relationship should be set to control the order in which multiple external files are appended to the requirement description when it is output in formal documentation generated from the database.

Table 8 States and Modes

Element Class	Attributes	Relationships	Target Classes
	Type	*included in / includes*	**State/Mode**
		includes / included in	**State/Mode**
		specified by / specifies	**Requirement**

3.2 Develop the System Functional Hierarchy

Functional analysis in CORE begins with defining major threads through the system and culminates in an integrated behavior model of the system and subcomponent actions. For the system, a top-level **Function** should be defined and *allocated to* the system. The *allocated to* relationship attribute Behavior Type should be set to "Integrated (Root)" in order to identify that this top-level **Function** represents the totality of functionality performed by the system and is decomposed (hierarchically) into all of the functions performed by the system. The root **Function** is decomposed into the primary **Functions** of the system.

Function Traceability. If a **Function** is identified in direct response to an originating **Requirement** or to an **Issue** decision, the **Function** should be linked to the causal elements, using either the *based on/basis of* or *result of/results in* relationships, in order to establish requirements traceability beyond the **Function** hierarchy. This also supports the use of **Functions** as requirements, i.e. the inclusion of "shall" in the **Function** Description.

State/Mode Mapping. If **State/Modes** have been defined (See Section 3.1), they should be linked to the first-level (i.e., non-root level) **Functions** to identify which **Functions** and their descendants are available in each **State/Mode**.

Function Allocation. In conjunction with Physical Architecture Synthesis (See Section 4.1), for each layer of **Components**, **Functions** are decomposed until they can be uniquely allocated to the next level of **Component**. These allocations are considered atomic, that is non-root. When generating formal documentation, this functional hierarchy and allocation provides the organization of the performance **Requirements** in a specification for a **Component** (i.e., **Function** *specified by* **Requirement**).

Function Inputs and Outputs. For each **Function** in the evolving functional hierarchy, input and output **Items** are identified and associated using the relationships: *input to/inputs*, *output from/outputs*, and *triggers/triggered by*. When **Functions** are allocated in conjunction with Physical Architecture Synthesis (See Section 4.1), these **Items** form part of the definition of the component interfaces (See Sections 4.2 and 4.3). As with **Functions**, **Items** should be aggregated to simplify presentation.

Note: *When doing behavior modeling, a root* **Function** *can be established for any* **Component** *and the behavior diagram built using the allocated* **Functions** *to define the full behavior of the* **Component** *from the* **Component's** *perspective rather than from the system's perspective. These lower-level root* **Functions** *do not appear in the system functional hierarchy, but act as tap points into the hierarchy. The formal documentation scripts use either root or atomic* **Functions**, *whichever allocation is present.*

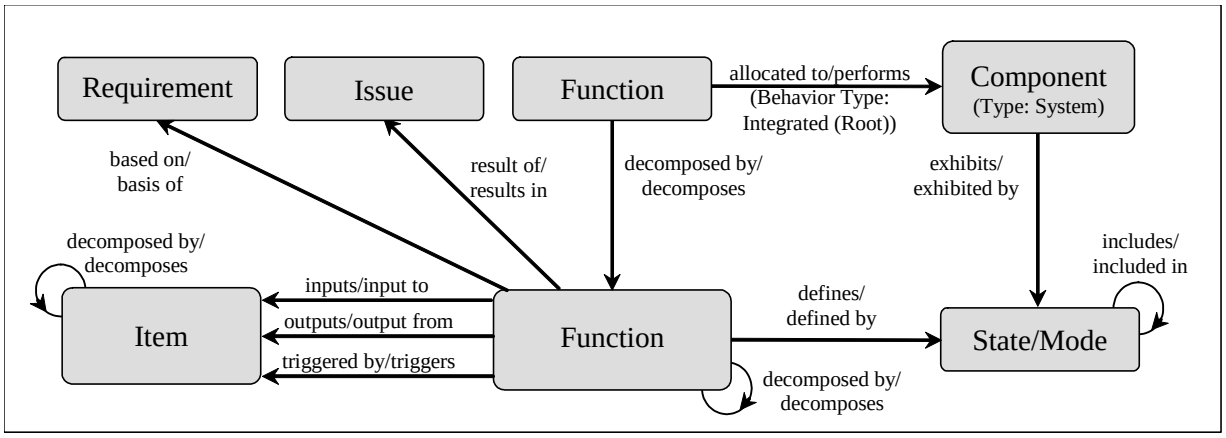

Figure 9 Functional Decomposition

Table 9 Functional Decomposition

Element Class	Attributes	Relationships	Target Classes
Component (Type: System)	See Section 1.1	*exhibits / exhibited by*	**State/Mode**
		performs / allocated to (Behavior Type: Integrated (Root))[8]	**Function**
Function	Description Doc. PUID Duration Number	*allocated to / performs* (Behavior Type: Integrated (Root))[5]	**Component**
		based on / basis of	**Requirement**
		decomposed by / decomposes	**Function**
		decomposes / decomposed by	**Function**
		defines / defined by	**State/Mode**[9]
		inputs / input to	**Item**
		outputs / output from	**Item**
		result of / results in	**Issue**
		triggered by / triggers	**Item**
Issue	See Section 2.2	*results in / result of*	**Function**

[8] A **Component** should have only one root **Function**.

[9] Only the top-level atomic **Functions** for the system should be linked to a **State/Mode**.

Table 9 Functional Decomposition

Element Class	Attributes	Relationships	Target Classes
Item	Accuracy Description Doc. PUID Number Precision Priority Range Size Size Units Type Units	*decomposed by / decomposes*	**Item**
		decomposes / decomposed by	**Item**
		input to / inputs	**Function**
		output from / outputs	**Function**
		triggers / triggered by	**Function**
Requirement	See Sections 1.2 and 2.1	*basis of / based on*	**Function**
State/Mode	See Section 3.1	*defined by / defines*	**Function**[6]
		exhibited by / exhibits	**Component**

3.3 Refine and Allocate Functional Performance Requirements

As the functional hierarchy is developed, decompose and allocate performance **Requirements** to **Functions**. This may be a complex process, particularly if it involves a domain change or trade studies that assimilate multiple performance **Requirements** to reach a design decision. The result of the design decision, captured as a **Requirement** whose Origin attribute is set to Design Decision, may result in multiple **Functions** and performance **Requirements** as well as constraint **Requirements**. If this is a major design decision, it should be augmented with an **Issue** to capture **Issue**-type information that is not normally captured by a **Requirement**.

Since **Functions** may be aggregated to enhance understanding, not every **Function** will have performance **Requirements**; however, **Functions** allocated to a **Component** should have performance **Requirements** to clearly define how well the functions must be performed in terms of such characteristics as timing and accuracy. Performance **Requirements** are inseparable from their associated **Functions**. Thus, only the **Function** is *allocated to* a **Component** (i.e., the performance **Requirement** for an allocated **Function** should not be linked with the *specifies* relation to the performing **Component**).

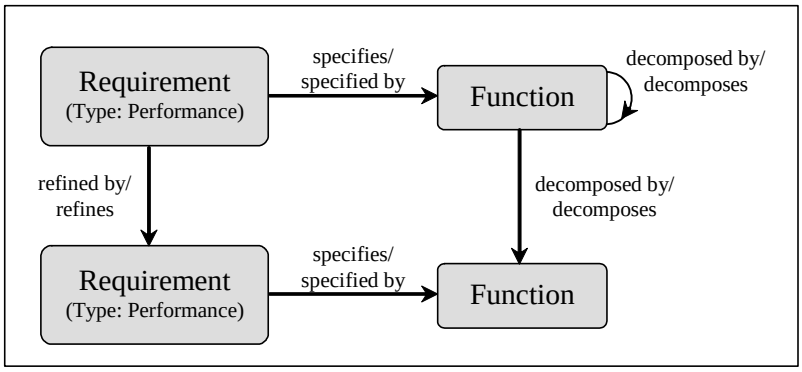

Figure 10 Performance Requirements

Table 10 Performance Requirements

Element Class	Attributes	Relationships	Target Classes
Function	See Section 3.2	*specified by / specifies*	**Requirement**
Requirement	See Sections 1.2 and 2.1	*refined by / refines*	**Requirement**
		refines / refined by	**Requirement**
		specifies / specified by	**Function**

3.4 Capture Functional and Performance Issues and Risks

While developing the system's functional hierarchy and deriving the associated performance requirements, additional issues and risks may be identified. They should be captured in the database in a manner analogous to **Issues** and **Risks** resulting from the analysis of originating requirements (See Section 2.2).

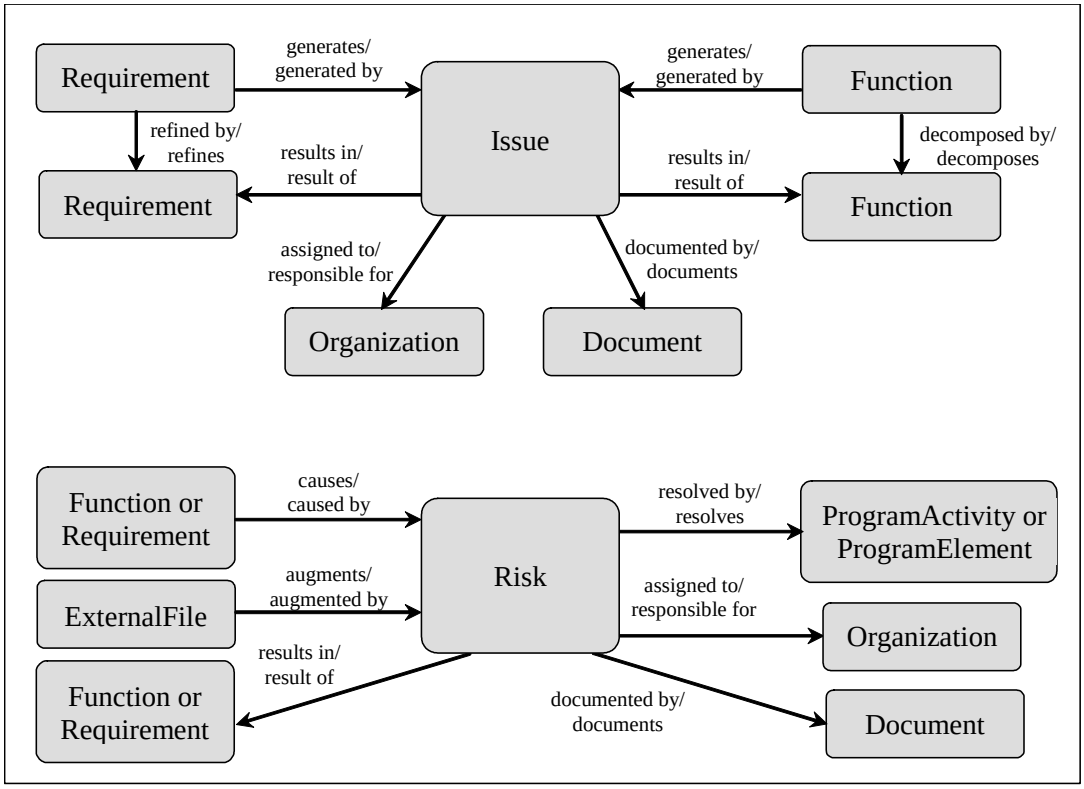

Figure 11 Functional or Performance Requirement Issue and Risk

Table 11 Functional or Performance Requirement Issue and Risk

Element Class	Attributes	Relationships	Target Classes
Document	See Section 2.2	*documents / documented by*	**Issue** **Risk**
ExternalFile	See Section 1.2	*documents / documented by*	**Issue** **Risk**
Function	See Section 3.2	*decomposed by / decomposes*	**Function**
		decomposes / decomposed by	**Function**
		causes / caused by	**Risk**
		generates / generated by	**Issue**

Table 11 Functional or Performance Requirement Issue and Risk

Element Class	Attributes	Relationships	Target Classes
		result of / results in	**Issue** **Risk**
Issue	See Section 2.2	*assigned to / responsible for*	**Organization**
		documented by / documents	**Document**
		generated by / generates	**Function** **Requirement**
		results in / result of	**Function** **Requirement**
Organization	See Section 2.2	*responsible for / assigned to*	**Issue** **Risk**
ProgramActivity	See Section 2.2	*resolves / resolved by*	**Risk**
ProgramElement	See Section 2.2	*resolves / resolved by*	**Risk**
Requirement	See Sections 1.2 and 2.1	*causes / caused by*	**Risk**
		generates / generated by	**Issue**
		refined by / refines	**Requirement**
		refines / refined by	**Requirement**
		result of / results in	**Issue** **Risk**
Risk	See Section 2.2	*assigned to / responsible for*	**Organization**
		augmented by / augments	**ExternalFile**
		caused by / causes	**Function** **Requirement**
		resolved by / resolves	**ProgramActivity** **ProgramElement**
		results in / result of	**Function** **Requirement**

4 Physical Architecture Synthesis

4.1 Allocate Functions to Next Level of Components

In conjunction with the analysis of requirements, functional decomposition, and assessment of component technology, identify the next layer of **Components** in the system component hierarchy.

As the component hierarchy evolves, **Functions** are uniquely *allocated to* **Components**. This allocation is done in layers. When a decomposed atomic **Function** is allocated to a **Component**, all lower-level **Functions** in its decomposition are part of the behavior of the **Component**. The **Component** may be decomposed, in which case even lower-level **Functions** are allocated to the lower-level **Components**. These allocations are termed Atomic. Since **Functions** can be aggregated to enhance understanding, there is not a one-to-one correspondence between levels in the **Function** hierarchy and levels in the **Component** hierarchy.

Note: *As stated in Section 3.2, when doing behavior modeling, a root* **Function** *can be established for any* **Component** *and the behavior diagram built using the allocated atomic* **Functions** *to define the full behavior of the* **Component** *from the* **Component's** *perspective rather than from the system perspective. These lower-level root* **Functions** *for lower-level* **Components** *do not appear in the system functional hierarchy, but act as tap points into the hierarchy.*

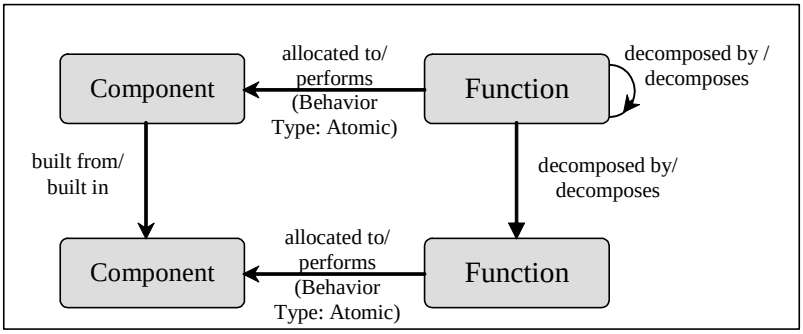

Figure 12 Component Hierarchy and Function Allocation

Table 12 Component Hierarchy and Function Allocation

Element Class	Attributes	Relationships	Target Classes
Component	Abbreviation Description Doc. PUID Purpose Number Type	*built from / built in*	**Component**
		built in / built from	**Component**
		performs / allocated to	**Function**
Function	See Section 3.2	*allocated to / performs*	**Component**

4.2 Refine External Interface Definitions

An external **Interface** element identifies the fact that the system communicates in some manner with an external **Component** (See Section 1.3). Details of the interface are captured in **Link** element definitions. As the system component hierarchy evolves, the terminus point for **Interfaces** and **Links** are changed, as appropriate, to lower-level **Components** when the components that provide the **Items** for the **Links** are determined by **Function** allocation. When the target of a *joins* or *connects to* relationship is changed from the system to one of its subordinates, CORE automatically establishes the *joined thru* or *connected thru* relationship between the **Interface** or **Link** and ancestors of the subordinate **Component** including the system. This allows **Interfaces** and **Links** to retain their identity even though their end points may change as the component hierarchy grows in depth. This also allows the content of interface and formal higher-level component specifications to remain unchanged as the **Interface**/**Link** connection points move deeper into the system component hierarchy.

Interfaces and **Links** may be *specified by* performance and constraint **Requirements**. Only the lowest layer of **Items** should be *transferred by* a **Link**.

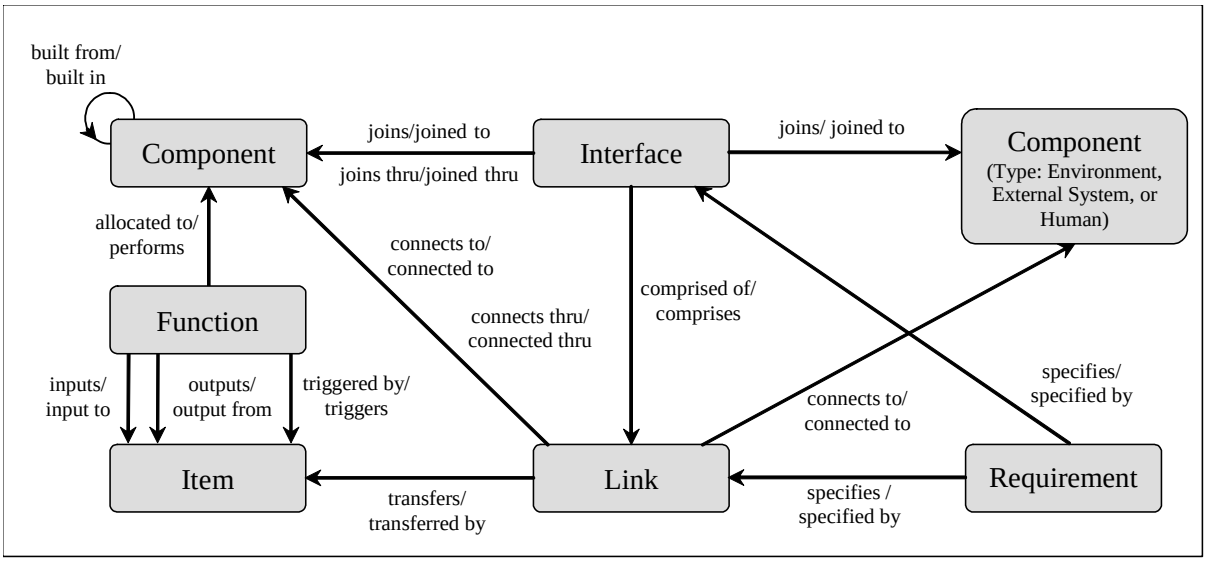

Figure 13 External Interface Definition

Table 13 External Interface Definition

Element Class	Attributes	Relationships	Target Classes
Component	See Section 4.1	*connected thru / connects thru*[10]	**Link**
		connected to / connects to	**Link**
		joined to / joins	**Interface**
		joined thru / joins thru[11]	**Interface**
Interface	See Section 1.3	*comprised of / comprises*	**Link**
		joins / joined to	**Component**
		joins thru / joined thru[12]	**Component**
		specified by / specifies	**Requirement**
Item	See Section 3.2	*transferred by / transfers*	**Link**
Link	Capacity Capacity Units Delay Delay Units Description Doc. PUID Number Protocol	*comprises / comprised of*	**Interface**
		connects thru / connected thru[13]	**Component**
		connects to / connected to	**Component**
		specified by / specifies	**Requirement**
		transfers / transferred by	**Item**
Requirement	See Section 2.1 Type: Performance or Constraint	*specifies / specified by*	**Interface** **Link**

[10] Automatically set based on the component hierarchy and *connected to* targets.
[11] Automatically set based on the component hierarchy and *joined to* targets.
[12] Automatically set based on the component hierarchy and *joins* targets.
[13] Automatically set based on the component hierarchy and *connects to* targets.

4.3 Derive or Refine Internal Interfaces

Within the system hierarchy, the allocation of **Functions** to **Components** establishes the internal interfaces of the system based on the **Items** that flow between the allocated **Functions**. The internal interfaces are formalized in the database using the **Interface** and **Link** element classes.

As the system component hierarchy evolves further, the terminus point for **Interfaces** and **Links** are changed, as appropriate, to lower-level **Components** when the components that provide the **Items** for the **Links** are determined by **Function** allocation. When the target of *joins* or *connects to* is changed from a **Component** to one of its subordinates, CORE automatically establishes the *joined thru* or *connected thru* relationship between the **Interface** or **Link** and the parent of the subordinate **Component**. This allows **Interfaces** and **Links** to retain their identity even though the end points may change as the system component hierarchy grows in depth. This also allows the content of formal higher-level specifications to remain unchanged as the **Interface**/**Link** connection points move deeper into the system component hierarchy.

Interfaces and **Links** may be *specified by* performance and constraint **Requirements**. Only the lowest layer of **Items** should be *transferred by* a **Link**.

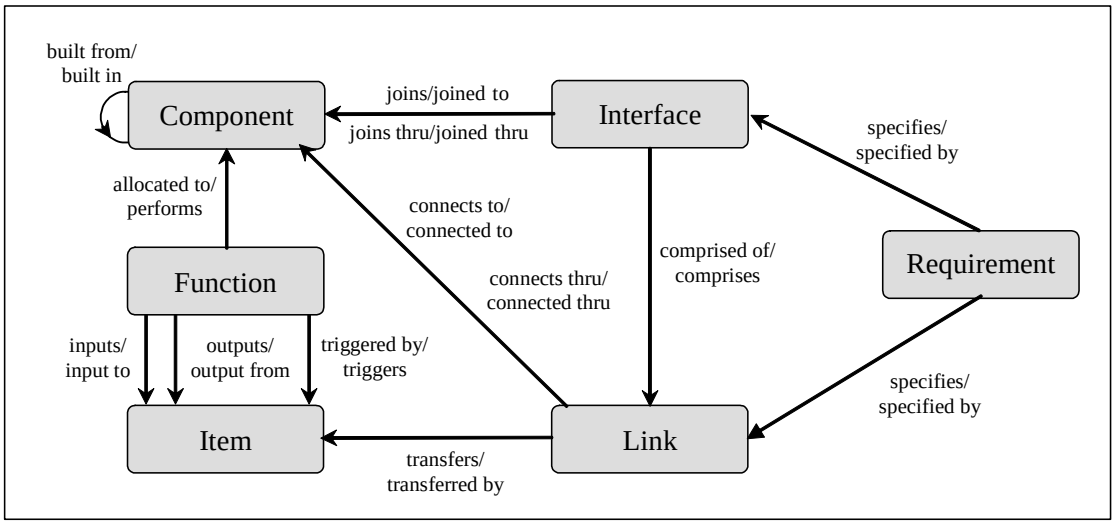

Figure 14 Internal Interface Definition

Table 14 Internal Interface Definition

Element Class	Attributes	Relationships	Target Classes
Component	See Section 4.1	*connected thru / connects thru*	**Link**[14]
		connected to / connects to	**Link**
		joined to / joins	**Interface**

[14] Automatically set based on the component hierarchy and *connected to* targets.

Table 14 Internal Interface Definition

Element Class	Attributes	Relationships	Target Classes
		joined thru / joins thru	**Interface**[15]
Interface	See Section 1.3	*comprised of / comprises*	**Link**
		joins / joined to	**Component**
		joins thru / joined thru	**Component**[16]
		specified by / specifies	**Requirement**
Item	See Section 3.2	*transferred by / transfers*	**Link**
Link	See Section 4.2	*comprises / comprised of*	**Interface**
		connects thru / connected thru	**Component**[17]
		connects to / connected to	**Component**
Requirement	See Section 2.1 Type: Performance or Constraint	*specifies / specified by*	**Interface** **Link**

4.4 Assign/Derive Constraints for Components

Based on the constraint **Requirements** allocated to a parent **Component**, constraint **Requirements** are derived for the subcomponents. This can be a simple flow-down of the same requirement or may be a budgeting of a limitation, such as weight, between subcomponents.

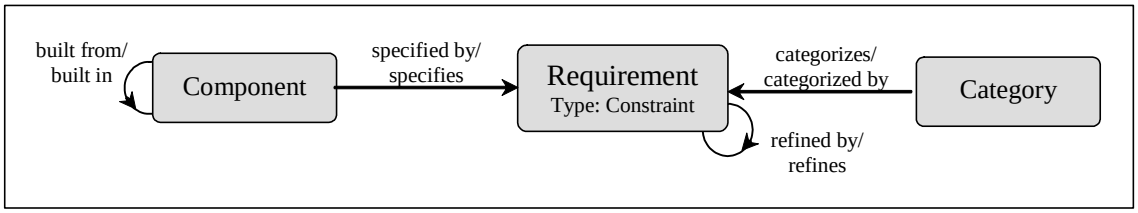

Figure 15 Component Constraint Requirements

[15] Automatically set based on the component hierarchy and *joined to* targets.
[16] Automatically set based on the component hierarchy and *joins* targets.
[17] Automatically set based on the component hierarchy and *connects to* targets.

Table 15 Component Constraint Requirements

Element Class	Attributes	Relationships	Target Classes
Category	Description	*categorizes / categorized by*	**Requirement** (Type: Constraint)
Component (Type: System)	See Section 1.1	*specified by / specifies*	**Requirement** (Type: Constraint)
Requirement (Type: Constraint)	See Section 1.2 Type: Constraint	*categorized by / categorizes*	**Category**
		refined by / refines	**Requirement** (Type: Constraint)
		refines / refined by	**Requirement** (Type: Constraint)
		specifies / specified by	**Component** (Type: System)

4.5 Capture Physical Architecture Issues and Risks

While developing the physical architecture and deriving **Interfaces** and performance/constraint **Requirements**, additional issues and risks may be identified. These should be captured in the database in a manner analogous to **Issues** and **Risks** resulting from the analysis of originating **Requirements** (See Section 2.2).

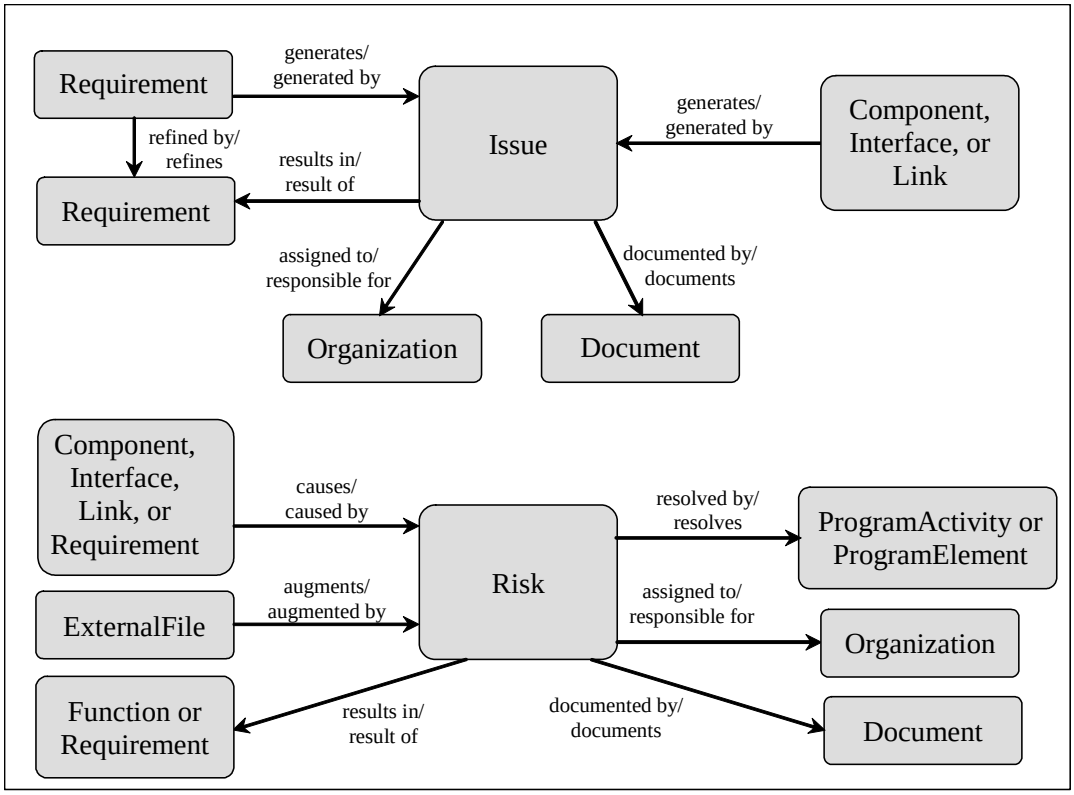

Figure 16 Physical Architecture Issue and Risk

Table 16 Physical Architecture Issue or Risk

Element Class	Attributes	Relationships	Target Classes
Component	See Section 4.1	*causes / caused by*	**Issue**
		generates / generated by	**Risk**
Document	See Section 2.2	*documents / documented by*	**Issue** **Risk**
ExternalFile	See Section 1.2	*documents / documented by*	**Issue** **Risk**
Function	See Section 3.2	*result of / results in*	**Issue** **Risk**
Interface	See Section 1.3	*causes / caused by*	**Risk**
		generates / generated by	**Issue**
Issue	See Section 2.2	*assigned to / responsible for*	**Organization**
		documented by / documents	**Document**

Table 16 Physical Architecture Issue or Risk

Element Class	Attributes	Relationships	Target Classes
		generated by / generates	**Component** **Interface** **Link** **Requirement**
		results in / result of	**Function** **Requirement**
Link	See Section 4.2	*causes / caused by*	**Risk**
		generates / generated by	**Issue**
Organization	See Section 2.2	*responsible for / assigned to*	**Issue** **Risk**
ProgramActivity	See Section 2.2	*resolves / resolved by*	**Risk**
ProgramElement	See Section 2.2	*resolves / resolved by*	**Risk**
Requirement	See Sections 1.2 and 2.1	*causes / caused by*	**Risk**
		generates / generated by	**Issue**
		refined by / refines	**Requirement**
		refines / refined by	**Requirement**
		result of / results in	**Issue** **Risk**
Risk	See Section 2.2	*assigned to / responsible for*	**Organization**
		augmented by / augments	**ExternalFile**
		caused by / causes	**Component** **Interface** **Link** **Requirement**
		resolved by / resolves	**ProgramActivity** **ProgramElement**
		results in / result of	**Function** **Requirement**

5 Verification/Validation

5.1 COREsim

COREsim is a discrete event simulator that executes the functional and link models to provide an assessment of system performance and to verify the dynamic integrity of the conceptual model. COREsim dynamically interprets a behavior model (i.e., the Enhanced Functional Flow Block Diagram (EFFBD)) in conjunction with the component link model and identifies and displays timing, resource utilization, link flow, and model inconsistencies. COREsim usage should be an integral part of functional analysis and physical architecture synthesis.

5.2 Establish Verification Requirements

For each specified **Component**, including the system, establish how each **Requirement** is to be verified. This information is captured in the database using **VerificationRequirements**. **VerificationRequirements** can range from requirements on acceptance testing such as qualification test to verification of individual **Requirements** and **Functions**. A single **VerificationRequirement** verifies multiple requirements.

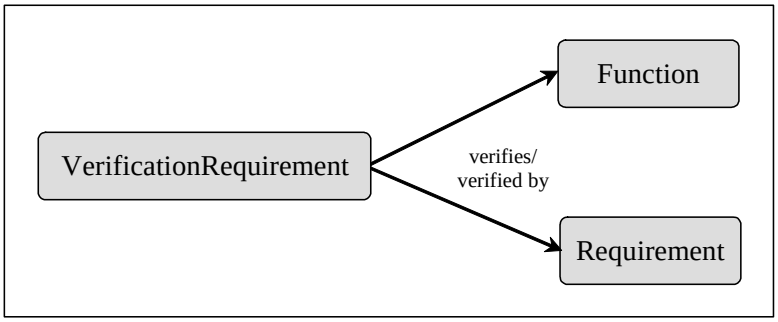

Figure 17 Verification Requirements

Table 17 Verification Requirements

Element Class	Attributes	Relationships	Target Classes
Function	See Section 3.2	*verified by / verifies*	**Verification Requirement**
Requirement	See Section 2.1	*verified by / verifies*	**Verification Requirement**
Verification Requirement	Description Doc. PUID Level Method Number	*verifies / verified by*	**Function** **Requirement**

5.3 Establish Verification Events and Test Procedures

Actual verification activities are captured in the database as **TestEvents**. **TestProcedures** identify the procedures. It is assumed that the actual test steps and expected results are external to the CORE database and are referenced in the database using either **Document** or **ExternalFile** elements. **TestConfigurations** identify the equipment needed for particular **TestEvents**. A **TestConfiguration** identifies **Components** of the system and **Links** to the **Components** as well as test equipment and software. As verification events are planned and conducted, the **VerificationRequirement's** Status attribute is updated in the database.

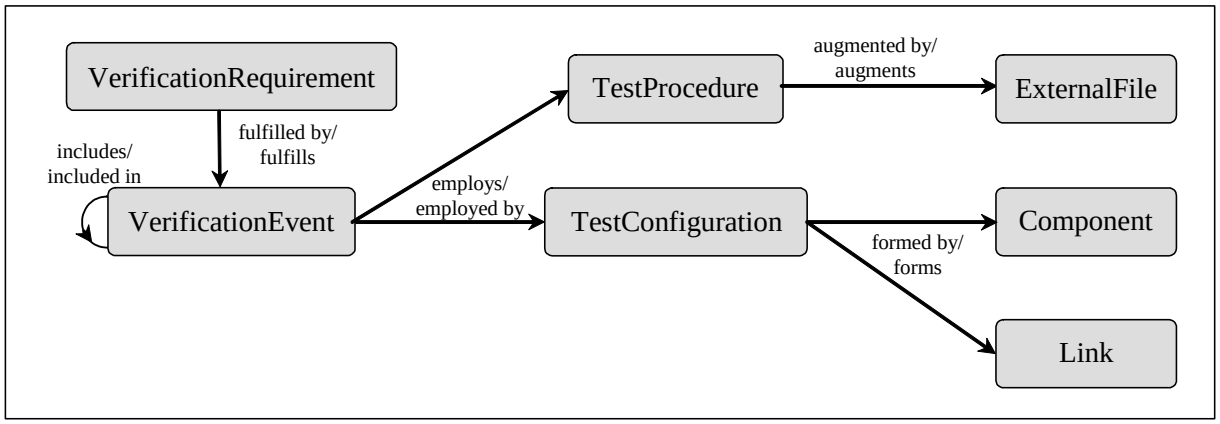

Figure 18 Verification Planning and Tracking

Table 18 Verification Planning and Tracking

Element Class	Attributes	Relationships	Target Classes
Component	See Section 4.1	*forms / formed by*	**TestConfiguration**
Document	Description External File Path Number	*documents / documented by*	**TestProcedure**
ExternalFile	See Section 1.2	*augments / augmented by*	**TestProcedure**
Link	See Section 4.1	*forms / formed by*	**TestConfiguration**
Organization	Description Number Role	*responsible for / assigned to*	**VerificationEvent**
TestConfiguration	Description Number	*formed by / forms*	**Component** **Link**
		employed by / employs	**VerificationEvent**
TestProcedure	Description	*employed by / employs*	**VerificationEvent**

Table 18 Verification Planning and Tracking

Element Class	Attributes	Relationships	Target Classes
	Number	*augmented by / augments*	**ExternalFile**
VerificationEvent	Description Duration Duration Units End Date Number Start Date	*assigned to / responsible for*	**Organization**
		employs / employed by	**TestConfiguration** **TestProcedure**
		fulfills / fulfilled by	**VerificationRequirement**
		included in / includes	**VerificationEvent**
		includes / included in	**VerificationEvent**
Verification Requirement	See Section 5.2	*fulfilled by / fulfills*	**VerificationEvent**

6 Formal Documentation

CORE's flexible approach to the generation of formal specifications and documents allows the generation of draft documents as soon as the project team begins populating the database as described in Sections 2 through 5. A library of section level scripts and pre-defined outlines for a System/Subsystem Specification (SSS), System/Subsystem Design Document (SSDD), Interface Requirements Specification (IRS), Software Requirements Specification (SRS), and Test & Evaluation Plan (TEP) is provided with CORE.

To take advantage of this capability, document-oriented element classes and relations are used to define a document's outline and control information, and to reference section content and scripts. The **Document**, **Section**, **ReportScript,** and **DocumentFormat** classes are key to document definition.

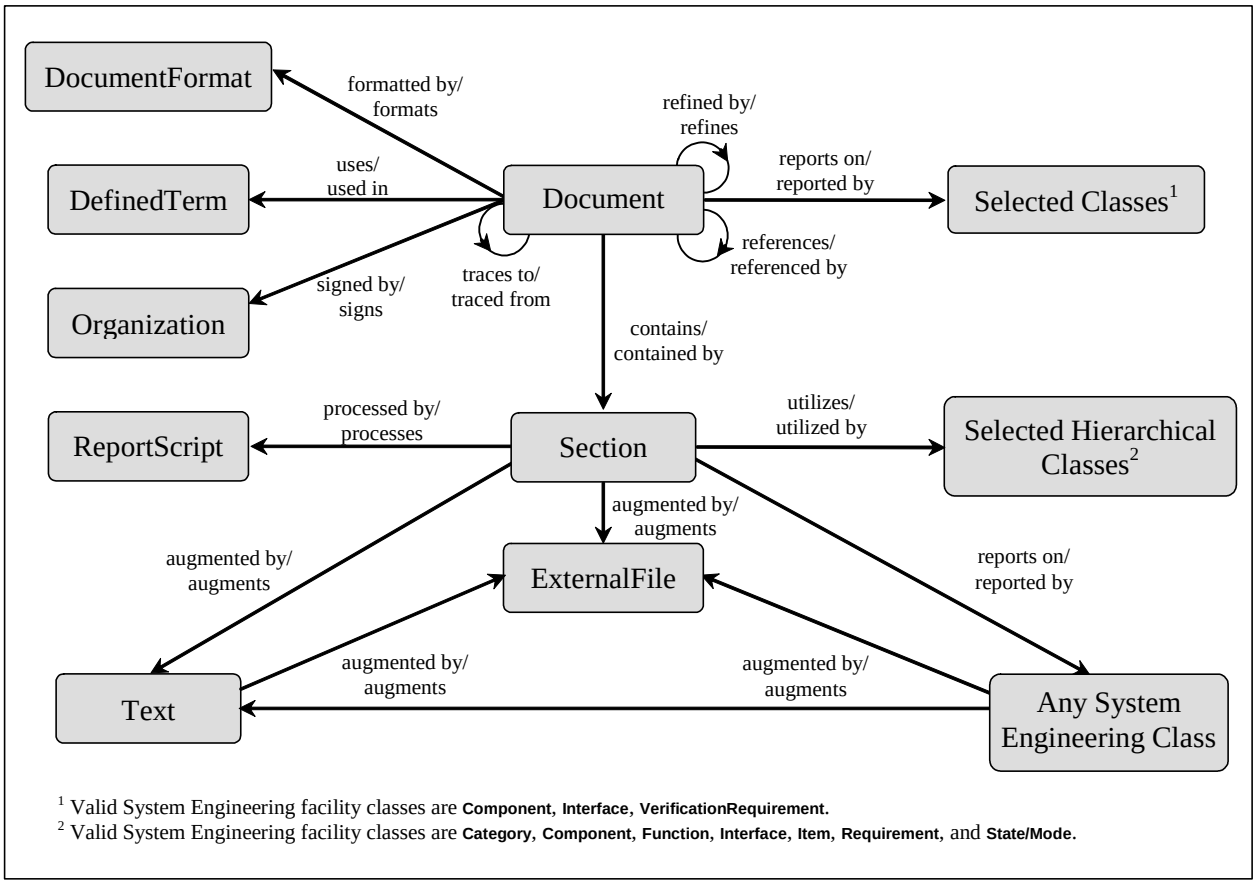

Figure 19 Formal Document Generation

Table 19 Formal Document Generation

Element Class	Attributes	Relationships	Target Classes
DefinedTerm	Acronym Description	*used in / uses*	**Document**
Document (generated from database)	CDRL Number Description Doc. Script Document Date Document Number External File Path either[18] Govt. Category or Non-Govt. Number Number Section Suffix Title Type	*contains / contained by*	**Section**
		referenced by / references[19]	**Document**
		references / referenced by[16]	**Document**
		refined by / refines[20]	**Document**
		refines / refined by[17]	**Document**
		traced from / traces to[21]	**Document**
		traces to / traced from[18]	**Document**
DocumentFormat	CAGE Code Contract Number Description Number PUID Placement Req. Format	*formats / formatted by*	**Document**
ExternalFile	Description External File Path Number Page Orientation Title Type	*augments / augmented by*	All System Engineering Facility Classes **Section** **Text**
Organization		*signs / signed by*	**Document**

[18] Required if **Document** is a reference/applicable document for another formal document generated from the database.

[19] Establishes contents of applicable/reference document section.

[20] Establishes documentation tree.

[21] Controls requirements traceability matrix.

Table 19 Formal Document Generation

Element Class	Attributes	Relationships	Target Classes
ReportScript	File Name Location Type	*processes / processed by*	**Section**
Section	Description Logical Diagram Type Req. Format Title	*augmented by / augments*	**ExternalFile** **Text**
		contained by / contains	**Document**
		processed by / processes	**ReportScript**
		reports on / reported by	All System Engineering Facility Classes
Text	Description	*augments / augmented by*	All System Engineering Facility Classes **Section**
		augmented by / augments	**ExternalFile**

This approach allows:

- System engineers to change a document's outline and, often, a section's content without scripting
- Reuse of section script output logic for different specifications/documents without scripting
- Rapid development of new documents with a minimum of scripting
- Document to document traceability in an efficient manner that allows documentation as well as engineering impact assessment
- Simplified reproducibility by capturing document output control information in the database.

In addition to the formal documentation scripts, document generation templates for the SSS, SSDD, IRS, SRS, and TEP scripts are provided with CORE. A document generation template is an RDT file that contains **Document**, **DocumentFormat**, **Section**, **ReportScript**, and **Category** elements with the appropriate attribute values and relationships to produce a document. The templates contain only **Section** elements for the non-database dependent sections of the outline. In addition to identifying section output scripts, the templates identify create subsection scripts that, when run for the parent **Section**, analyze the database and create and link subsections to complete the document outline.

The steps to produce a document using the standard document generation templates are the following:

1. Import the selected document generation template.
2. As needed, link the template to elements in the database following the script instructions.
3. Run the Create Subsections script.
4. Run the selected specification/document output report script.

Details of the database set-up, script execution, and template contents can be found in the formal documentation scripts help file. This can be opened by selecting the Run Script icon on the CORE Explorer toolbar, selecting any one of the formal documentation scripts, and pressing the Script Help File button.

The document generation templates allow a spectrum of customization options. Customization could be as simple as changing a section title or number or it could consist of assigning different section scripts. Full customization could include defining a new template and generating new section scripts.

In addition to the formal documentation scripts, a System Description Document (SDD) script is provided that produces an engineering view of the database. Also provided are numerous engineering support scripts such as the Generic Table Output, Indented Hierarchy Report, Element Definition, HTML Report, et al. These should be used on an on-going basis to aid the system engineers in communication and assessment of the system definition.

Vitech Corporation

2070 Chain Bridge Road, Suite 100
Vienna, Virginia 22182-2536
703.883.2270 FAX: 703.883.1860
Customer Support: support@vitechcorp.com
www.vitechcorp.com